(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
31 January 2002 (31.01.2002)

PCT

(10) International Publication Number
WO 02/09384 A2

(51) International Patent Classification[7]: H04L 29/00

(21) International Application Number: PCT/EP01/07898

(22) International Filing Date: 9 July 2001 (09.07.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/624,648 25 July 2000 (25.07.2000) US

(71) Applicant: **KONINKLIJKE PHILIPS ELECTRONICS N.V.** [NL/NL]; Groenewoudseweg 1, NL-5621 BA Eindhoven (NL).

(72) Inventor: **MOONEN, Jan, R.**; Prof. Holstlaan 6, NL-5656 AA Eindhoven (NL).

(74) Agent: **GROENENDAAL, Antonius, W., M.**; Internationaal Octrooibureau B.V., Prof. Holstlaan 6, NL-5656 AA Eindhoven (NL).

(81) Designated States *(national)*: CN, JP, KR.

(84) Designated States *(regional)*: European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 02/09384 A2

(54) Title: UI-BASED HOME NETWORK BRIDGING

(57) Abstract: A home network comprises a UPnP cluster and a HAVi cluster. UPnP uses programmatic device interfaces that are based on standardized messages being sent between the devices. HAVi also uses programmatic interfaces but needs to know the proper device type and FCMs in advance. In addition, the current UPnP and HAVi standards do not define devices that can readily be mapped onto one another owing to semantic differences. To overcome this problem, the clusters are bridged by representing a UPnP device on the HAVi cluster, wherein the UPnP device's description document is used to generate a HAVi DDI target to enable UI-based control of UPnP devices through a HAVi UI.

UI-based home network bridging

FIELD OF THE INVENTION

The invention relates to home networks, and especially to bridging networks that have clusters with different software architectures.

BACKGROUND ART

It seems to be unlikely that there will ever be a single universally applicable networking standard for each device in the home. Multiple standards for software architectures coexist and new ones will emerge. New standard interfaces will be developed for new types of devices that are specifically targeted by those standards. Home applications are designed to make an intelligent use of all devices available in the home, but are not capable of dealing with each and every networking standard in use or becoming available in the future. Similarly, devices themselves will not be able to support every existing home networking standard. For these reasons, bridges are needed between the different sub-networks or clusters, each respective one complying with a specific respective standard. A bridge serves to transparently represent a device, complying with a first standard in a network cluster of a first type, as a device complying with a second standard in a network cluster of a second type. The result is a single unified view of the home network available to software applications written for the first standard and as well those written for the second standard.

Home networking standards generally address either the lower levels of the protocol stack (up to transport level, e.g., HomePNA, HomeRF) or the higher levels (from transport level to application level). In the remainder of this description reference is made in general to the second type of home networking standards. Examples of these standards are HAVi, UPnP and Jini.

Typically, a home networking standard offers one of two ways for applications (or clients) to control devices (or servers): programmatic control and UI (user-interface)-based control.

Programmatic control refers to control that is based on sending standardized messages to a device, or control that is based on calling a standardized API, resulting in messages sent to a device. This kind of control is useful for applications that control devices

without any direct user-interaction being required. In this case, both the controller and the controllable device use a pre-determined set of messages that can be exchanged. That is, the controller has to know the type of controllable device in advance.

UI-based control refers to control that uses sending user interactions from a controlling device to a controllable device. Here, the user interacts with a UI client or browser, and user actions lead to messages sent to a UI server on the controllable device. The messages are exchanged using a particular UI protocol, examples of which are HTML (used in UPnP) and DDI (used in HAVi). The acronym "DDI" stands for Data Driven Interaction and refers to a protocol that is executed between applications or DCMs, whose User Interface needs to be displayed, on one side and a display device on the other. The DDI target is part of a device's DCM. The acronym "DCM" stands for Device Control Module. A DCM is a software element that represents a single device or functionality on the HAVi network. The DCM exposes the HAVi defined APIs for that device. DCMs are dynamic in nature: if a device is inserted or removed from the network, a DCM for that device needs to be installed or removed, respectively, in the network. DCMs are central to the HAVi concept and the source of flexibility in accommodating new devices and features into the HAVi network. The DDI supports user interaction with the appliance.

Essential is that the UI protocol does not define the kind of information or control conveyed by it, it merely defines how to encapsulate user actions and send them from client to server. Additionally, it defines how user interface elements (also called widgets) are described and sent back from server to client for display purposes. This implies that UI-based control is possible even when the controller application does not know the type of controllable device.

As long as controller and controllable device communicate using the same UI protocol, and a user is present, the controllable device can be controlled. Additionally, a UI protocol enables a device to present features that go beyond the standardized device type capabilities, in other words, it enables a standardized device to distinguish itself from the competition.

SUMMARY OF THE INVENTION

When bridging control between home networking standards, the natural approach is to translate between the programmatic interface of a device type in a first software architecture of standard A to the programmatic interface of the device type in a second software architecture in a standard B different from standard A, and vice versa.

Similarly, a natural approach is to translate the UI interface of a device type in standard A to the UI interface of the device type in standard B, and vice versa. The inventor has realized that certain problems are associated with this approach and has analyzed the bridging in further detail so as to provide a solution.

Fig.1 is a diagram to clarify that there is more than these options. The diagram of Fig.1 shows part of a home network 100 that has a cluster 102 with a software architecture complying with a standard A, and a cluster 104 with a software architecture of a standard B. Network 100 has a device 106 of a certain type X residing on network 100 in cluster 102. Clusters 102 and 104 are to be bridged so that device 106 can be controlled from cluster 104 through a software representation 108. Assume that device 106 and its representation 108 in cluster 104 in principle can have UI-based control interfaces 110 and 112, respectively, and programmatic control interfaces 114 and 116, respectively. The diagram lists four bridging options 118, 120, 122 and 124.

Bridge 118 provides a translation from UI interface 110 to programmatic interface 116. As described earlier, a UI interface is much more flexible (or less standardized) than a programmatic interface, so translating from a UI interface to a programmatic interface is very difficult, and, for existing home networking standards such as HAVi and UPnP, impossible. This leaves three other options:

- A bridge 120 for a translation of UI interface 110 in standard A to UI interface 112 in standard B. Whether this is feasible or not depends on the similarity of standards A and B. For example, translating a UPnP device UI to a HAVi device UI is very difficult, since UPnP allows HTML plus any kind of scripting in a device UI. HAVi uses the DDI protocol for device UI, and while HTML is probably translatable to DDI, translating HTML plus scripting is probably not feasible.

- A bridge 122 for a translation of programmatic interface 114 in standard A to programmatic interface 116 in standard B. Whether this is possible or not depends largely on the domains covered by standards A and B. For example, if UPnP defines a programmatic interface of a light bulb, and HAVi does not define a 'semantically equivalent' programmatic interface of a light bulb, then this device type cannot be bridged in a programmatic way.

- A bridge 124 for a translation of programmatic interface 114 in standard A to UI interface 112 in standard B. This type of translation requires that the programmatic interface of devices in standard A be available, at run-time, as data for translation purposes. This requires a 'white-box' device description mechanism, where an application can retrieve, at run time, all the information about the device's interface. In other words, an application

does not need to rely on built-in knowledge of the interface of the device type, it can retrieve this knowledge at run-time. This type of translation further requires that the data types used in the programmatic interface be mapped as 'modally compatible' GUI elements. Within this context, see co-pending U.S. serial no. 09/165,682 (attorney docket PHA 23,484) filed 10/2/98 for Eugene Shteyn for CONTROL PROPERTY IS MAPPED ONTO MODALLY COMPATIBLE GUI ELEMENT. The translation of a programmatic interface in standard A to a UI in standard B further requires that the set of modally compatible GUI elements be supported by the UI mechanism of standard B. It further requires that the abstraction level of the programmatic interface be suitable for user-level interaction.

All of the above translations may coexist in a bridge implementation. In fact, a bridge should represent a device in the other network in as much ways as possible, since it does not know which interface is most suitable for the applications/users on the other side of the bridge.

The current invention is based on the insight that the translation of a programmatic interface in a standard A to a UI in standard B can be used as a bridge. Below, the requirements are discussed posed on both standards involved for this bridging to work. Embodiments are given below of the bridging option in the form of a translation scheme for a UPnP programmatic interface to a HAVi device UI (DDI target).

The invention relates to a method of enabling to bridge a first cluster of first functionalities and a second cluster of second functionalities in a home network. The first cluster has a first software architecture that provides control of the first functionalities through a programmatic interface. UPnP is an example of such an architecture. The second cluster has a second software architecture that provides control of the second functionalities through a UI-based interface. HAVi is an example of the latter architecture. The method in the invention comprises providing a translation of the programmatic interface to the UI-based interface. In the UPnP - HAVi example, the method comprises generating a HAVi DDI target software element from a UPnP device description document.

The method of the invention can be implemented by a service provider. The user notifies the service provider of the new devices added to his/her network, whereupon the provider can generate the translation module to be installed on the user's network as part of a bridge.

The invention also relates to a home network with these clusters using different software architectures. The first cluster has a first software architecture that provides control of the first functionalities through a programmatic interface, and the second

cluster has a second software architecture that provides control of the second functionalities through a UI-based interface. The network has a bridge between the first and second clusters for translating the programmatic interface to the UI-based interface. The network preferably comprises a generator for generating a HAVi DDI target from a UPnP device description document.

An aspect of the invention resides in bridging software for use on a home network, wherein the network comprises a first cluster of first functionalities and a second cluster of second functionalities. The first cluster has a first software architecture that provides control of the first functionalities through a programmatic interface, and the second cluster having a second software architecture that provides control of the second functionalities through a UI-based interface. The bridging software is operative to couple the first and second clusters based on translating the programmatic interface to the UI-based interface. The software comprises a generator for generating a HAVi DDI target from a UPnP device description document in the UPnP - HAVi network.

BRIEF DESCRIPTION OF THE DRAWING

The invention is explained in further detail, by way of example and with reference to the accompanying drawing, wherein:

Fig.1 is a block diagram illustrating the options for bridging;

Figs.2A and 2B are a table with UPnP and HAVi information items referred to throughout the explanation below;

Fig.3 is a diagram with a portion of a home network;

Figs. 4, 5 and 6 are diagrams illustrating interactions between UPnP and HAVi clusters in a home network.

Throughout the Figures, same reference numerals indicate similar or corresponding features.

DETAILED EMBODIMENTS

The UPnP standard defines the programmatic interface of a device through an XML document called the "description document". An application can retrieve this document, through HTTP, at run-time, so UPnP satisfies the 'white-box' criterion.

HAVi defines the programmatic interface of a device through publication of an API expressed in IDL. The API definition is tied to a unique identifier called the FCM type. The acronym "FCM" stands for Functional Component Module. A DCM (see above)

contains an FCM for each controllable function within the represented device. Currently, HAVi defines FCM's and corresponding API's for functions such as a tuner, VCR, HDD-based storage, AV display, camera's and modems. At run-time a HAVi application can retrieve this FCM type from a controlled device, and is able to control the device programmatically based upon its built-in knowledge about the interface linked to the FCM type. Since the HAVi application needs to know the interface linked to the FCM type in advance, HAVi does not satisfy the 'white box' criterion.

In Jini, an application retrieves the programmatic interface of a device by using a lookup service that returns a remote (RMI) reference to the device's Java object representation. Java RMI (Remote Method Invocation) is used to issue control commands to the controlled device. Since Java has facilities for 'reflection', it is possible to determine, at run time, the interface (in terms of Java member variables and methods) of a remote object (device) reference. Hence, Java satisfies the 'white-box' criterion.

The programmatic interface of a UPnP device may use any of the following types: Boolean, Integer (called i4), Floating point (called r8), String, DateTime (data and time information) or HEX-encoding binary data (called bin.hex or bin.bin64). These types can be mapped in a straightforward manner to common GUI elements found in most UI protocols, including DDI (used in HAVi) and HTML (used in UPnP). See Figs. 2A and 2B. Jini does not define a specific device UI protocol.

The programmatic interface of a Jini device may use any Java object. The programmatic interface of a HAVi device may use any IDL type. Hence, both HAVi and Jini device interfaces can, in general, not easily be mapped to a UI. Of course, one can imagine that for a particular device type that uses a subset of the rich data type features of IDL or Java, a UI mapping is possible. For example, for a Jini interface that uses only basic Java types (no Java objects) a UI mapping is feasible.

In summary, it seems most valuable to define a generic translation of the programmatic interface of a UPnP device to a DDI UI in HAVi.

Translating a UPnP programmatic interface to a HAVi DDI UI

The programmatic interface of a UPnP device is described in a *description* document. This document specifies a *root* device with zero or more *(sub) devices*. Each device may have another list of sub devices, etc. A device without any sub devices (a 'leaf' in the tree) contains a single *service*. A service consists of a list of state variables and a list of actions. Services don't have sub services. State variables have a 'name' field, 'type' field, and

optionally fields to restrict the range of the type. Actions have a 'name' field and a list of parameters. Each parameter is described by a 'name' field, and a field called 'relatedStateVariable'. This field has to refer to a state variable name, and it indirectly defines the type of the action parameter.

An aspect of this invention is to generate a HAVi DDI Target from a UPnP device description document. The translation concerns both static aspects of the DDI Target as well as dynamic aspects.

As to static aspects: DDI elements or widgets need to be defined for UPnP data type; the organization of all DDI elements corresponding to a UPnP service needs to defined; the organization of all DDI elements corresponding to a UPnP device needs to defined.

As to dynamic aspects: DDI navigation between devices and services needs to be defined; initialization and termination of a 'session' between application and device needs to be defined;
the effect of asynchronous changes at the controlled UPnP device needs to be defined; the effect of user actions that are forwarded to the generated DDI target needs to be defined.

These aspects are described in further detail below.

Static: Translating UPnP data types to HAVi DDI elements

UPnP data types occur in two ways in a service description, namely, directly: as type of a state variable, and indirectly: through reference of a <relatedStateVariable> for an action parameter.

Some DDI elements are 'interactive': they allow a user to interact with them. For example, a button is interactive, while a text label is not interactive. Depending on the context, some interactive DDI elements may be 'temporarily' non-interactive or disabled. To express this, interactive DDI elements have an 'interactivity' field. Since UPnP allows state changes through actions only, DDI elements representing action parameters will set this field to ENABLED. DDI elements representing the current value of a device's state variable, however, will have this field set to DISABLED.

Some UPnP data types may have additional specifiers that indicate a particular restriction on the range of value that it may hold. This can could be used to generate more appropriate DDI elements, in some cases.

Static: Translating a UPnP service to a DDI structure

A service consists of a list of state variables and a list of actions. State variables have a 'name' field, 'type' field, and optionally fields to restrict the range of the type. Actions have a 'name' field and a list of parameters. Each parameter is described by a 'name' field, and a field called 'relatedStateVariable'. This field has to refer to a state variable name, and it indirectly defines the type of the action parameter.

In DDI, elements may be organized in a group, in order to suggest some sort of coupling to a DDI display engine (called Ddi controller). This coupling may be used to determine the organization of elements on the screen, or to determine navigation. A UPnP service can be translated to the following DDI structure:

- A DdiPanel element with panelName set to the <serviceType> field of the service. The DdiPanel contains two DdiGroup elements and a DdiPanelLink element:

- A DdiGroup element with groupName set to "State Variables". The 'elements' field of this group should contain DDI elements obtained from translating all UPnP state variables according to the type mapping described earlier. All of these Ddi elements should have their Interactivity attribute set to DISABLED.

- A DdiGroup element with groupName set to "Actions". The 'elements' field of this group contains DdiGroup elements for each individual UPnP action. Each of these DdiGroup elements contains:

- A DdiButton element which both 'pressedLabel' and 'releasedLabel' fields set to the <name> field of the action. This element represents invoking the action on the UPnP device.

- Ddi elements for each input parameter of the action, according to the type mapping applied to the <relatedStateVariable> field of the action parameter. All of these Ddi elements should have their Interactivity attribute set to DISABLED.

- Ddi elements for each output parameter of the action, according to the type mapping applied to the <relatedStateVariable> field of the action parameter. All of these Ddi elements should have their Interactivity attribute set to ENABLED.

- A DdiPanelLink element representing the 'parent' device will its linkName field set to the parent device's <deviceType> or <friendlyName> field. Optionally, it could have the linkBitMap field set to one of the available and HAVi-compatible <icon> values in the parent device description.

Static: Translating a UPnP device to a DDI structure

A device description document in UPnP specifies a *root* device with zero or more *(sub) devices*. Each device may have another list of sub devices, etc. Additionally, each device has certain fields associated with this, such as a device icon, a manufacturer name, etc.. A (root or sub) device can be translated to the following DDI structure:

- A DdiPanel element with panelName set to the <deviceType> or <friendlyName> field of the device. The DdiPanel contains DdiPanelLink elements, one for each sub device or service that it contains.

- A DdiPanelLink element representing a sub device should have its linkName field set to the sub device's <deviceType> or <friendlyName> field. Optionally, it could have the linkBitMap field set to one of the available and HAVi-compatible <icon> values in the device description.

- A DdiPanelLink element representing a service should have its linkName field set to the service's <serviceType> field.

Each non-root sub device should have:

- A DdiPanelLink element representing the 'parent' device will its linkName field set to the parent device's <deviceType> or <friendlyName> field. Optionally, it could have the linkBitMap field set to one of the available and compatible <icon> values in the device description.

All DdiPanelLinks should have the Interactivity field set to ENABLED. Optionally, the panel could contains DdiText elements, with the Interactivity field set to DISABLED and the textContent field holding the string value for any of the following UPnP device description fields:

<deviceType>;
<friendlyName>;
<modelDescription>;
<modelName>;
<modelNumber>;
<modelURL>;
<manufacturer>;
<manufacturerURL>;
<serialNumber>;
<UDN>

Finally, it could have a DdiIcon element set to one of the available and compatible <icon> values in the device description.

An example UI that might be rendered by a DDI controller from a DDI target structured this way is shown in Fig. 3.

Fig.3 is a diagram of part 300 of home network 100. Network 100 has a root device 302: a TV/VCR combo, and two sub devices: a TV 304 and a VCR 306 . TV 304 has three services: an Audio service 308, a Video service 310 Video and a Tuner service 312. VCR 306 has also three services: a Clock service 314, a Tuner service 316 and a Tape service 318. The example shows the use of the <friendlyName> field ("My BedRoom Fun"), the <manufacturer> field ("Philips"), the <modelName> field ("AZ1010"), the <modelDescription> field ("19" TV/VCR combo") and the <icon> field (TV picture). Tape service 318 shows a string state variable 320 named "Transport" with <allowedValueList> specifier: { "playing", "stopped", "recording" }. Furthermore, it shows three actions: "Play", "Stop" and "Rec", where the "Rec" action has a single string parameter related to a state variable (not shown here) with <allowedValueList> specifier: { "standard", "long", "extended" }.

Dynamic: Navigating the device UI

As explained above, each (root or sub) device and each service is mapped to a separate DdiPanel element. DdiPanelLink elements are used to traverse from the root device down to services via zero or more intermediate sub devices. Additionally, on each level, except on the root device level, a DdiPanelLink is used to travel up in the device hierarchy. When a user on the DDI controller device clicks on the panel, a user action of type ACT_SELECTED will be sent to the DDI target. The DDI target shall respond to this by returning the targetPanel of representing the sub device/service (in case of navigating down the hierarchy) or the parent device (in case of navigating up the hierarchy). The DDI target does not need to communicate with the UPnP device that it represents.

Dynamic: Initializing/Terminating a session

A session between a DDI controller and generated DDI target is shown in Fig.4. After a DDI target receives a DDI subscription in step 402 it retrieves the UPnP description document in step 404. Based on the description document the DDI target is able to construct in step 406 the DdiPanel structure as well as all DdiElements it contains. Additionally, the DDI target subscribes to UPnP device state changes, in step 408, using the

GENA mechanism. "GENAA" stands for Generic Eventing Notification within UPnP context. Eventing is the ability of a source device to initiate a connection at any time to one or more other devices that want to receive events from the source device. Events are used to establish synchronization among multiple devices. Within UPnP, events are mainly used for asynchronous notification of state changes. TCP/IP provides the support for connections to carry event information. GENA adds conventions for establishing relationships between interested devices and an addressing scheme to allow delivery of the event messages. GENA uses HTTP addressing and encapsulation. Separate subscriptions in step 410 are needed for each service. Since a GENA SUBSCRIBE returns the current values of all state variables, the DDI target is 'synchronized' with the UPnP device and is able to return the correct values for all DDI elements when the DDI controller requests them. A DDI target may subscribe to state changes immediately after the description document is retrieved (shown above) or in a 'just-in-time' fashion, just before the DDI controller requests the panel containing the state variable values. After a DDI target receives a DDI unsubscription in step 412 it unsubscribes itself in step 414 from state variable changes.

Dynamic: Asynchronous changes at the UPnP device

When a session has been established between a DDI controller and a DDI target, it is still possible that the device state changes in an asynchronous way through some other control. This might be another HAVi or UPnP application, or even the user pressing some physical button on the physical device's manual control panel. This scenario is schematically shown in Fig.5.

Dynamic: User Actions at the DDI controller

Fig.6 shows the message that results when a user interacts with the UPnP device, through the display of the device running the DDI controller. The mode of operation is that the user first modifies the widgets corresponding to the action parameters and, when the user is done, presses the button that invokes the action on the device. The result of the action, including any output parameter, is shown using widgets that are in 'disabled' or 'non-interactive' mode.

Fig.6 shows that typically a user first modifies the widgets corresponding to the action parameters. These lead to "UserAction' messages sent to the DDI target. The target will not forward these interactions to the UPnP device, but rather use them to locally update the parameter values, represented by the widgets. When the user is done, he/she presses the

button that represents the action on the device. On the "RELEASE' event of the DdiButton, the DDI target will construct a SOAP (Simple Object Access Protocol) request using the current parameter values, and sent it to the UPnP device. The SOAP action is executed synchronously, and when the action is executed successfully, the DDI Target will complete the user action by returning a DDI "change report" for the output parameter widgets, if there are any. The output parameter values are extracted from the SOAP response message.

As to SOAP, this is a protocol created with contributions from many industrial partners and that allows applications to communicate with each other over the Internet, providing a framework for, e.g., connecting Web sites and applications to create Web services. Web services link sites and applications together to perform functions that the individual components alone are not able to perform.

Typically, the action invoked on the controlled device will also change the state of the device and a GENA NOTIFICATION message will be sent, some time later, to the DDI Target indicating new values for all changed state variables. This in turn will lead to NotfyDdiChange messages, as described earlier, and ultimately to some visual change on the display of the DDI controller device.

Fig.6 illustrates the situation wherein there is no error, that is, the SOAP response that eventually gets issued by the UPnP device has return code OK. In case of an error, the DdiTarget can use the AlertPanel facility in DDI to indicate failure to the user. The 'alert panel' can contains a DdiText element holding the SOAP response <faultString> field. This is shown in Fig.7.

U.S. serial no. 09/165,682 (attorney docket PHA 23,484) filed 10/2/98 for Eugene Shteyn for CONTROL PROPERTY IS MAPPED ONTO MODALLY COMPATIBLE GUI ELEMENT relates to a home network system comprising an electronic device and a controller coupled to the device. The device exposes an abstract representation of its functionality to the controller. The controller enables controlling the device's functionality through interaction with the abstract representation. The abstract representation specifies a modality of controlling the functionality. Under control of the modality specified, the system associates the controlling of the functionality with a modally compatible controlling capability of the controller. The modality exposed can be, for example, "Boolean", "float", "integer array".

The term "modality" refers to an attribute that denotes the character of the controllability of the functionality. In the invention, the modality of the functionality is mapped onto a modaly compatible capability of the controller without the controller actually

having to know what the functionality of the device is all about. For example, assume that the modality is semantically a Boolean. The Boolean control property can then be mapped onto a UI element that has a Boolean character and assumes one of two states such as an "on/off" switch or "high/low" lever. When the user then interacts with this element, the functionality mapped onto it is put into one of the two states. In a HAVi system, the abstract representation gets uploaded, preferably including a UI (for, e.g., voice control) or GUI, and the user receives a context to determine the consequences of his/her interactions with the UI. In case the modality is a set of discrete values, the control property can be mapped onto a GUI element that can assume one of three or more discrete states such as a dial for channel selection on a device, for selection among multiple devices, etc. If the specified modality is semantically a range of floating-point values, a mapping is feasible onto a continuously variable control feature, e.g., brightness of an image on a display or sound volume. In another example, the specified modality is semantically an array. An array comprises more than a single component. For example, an array of Booleans could thus be mapped onto a cluster of GUI elements to implement, e.g., a menu selection among different devices or a check box list. An array of floating point modalities could be mapped onto a cluster of GUI elements that represent sliders, e.g., for adjusting a camera angles and zoom factors of camera's in the home security system that is controlled via the home network. Note that the controller does not need to have a clue about the functionality or functionalities of the device being controlled. All it needs to do is subjecting a functionality to a control application based on the semantics of its modality.

U.S. serial no. 09/340,272 (attorney docket PHA 23,634) filed 6/25/99 for Eugene Stheyn for BRIDGING MULTIPLE HOME NETWORK SOFTWARE ARCHITECTURES relates to the bridging of home networks of different software architectures. References to software representations of devices and services on a first one of the networks are automatically created. The references are semantically sufficient to enable automatic creation of at least partly functionally equivalent software representations for a second one of the networks so as to make the devices and services of the first network accessible from the second network. This document also discusses some aspects of HAVi in further detail. This document also addresses the HAVi, Home API and Jini software architectures.

U.S. serial no. 09/160,490 (attorney docket PHA 23,500) filed 9/25/98 for Adrian Turner et al., for CUSTOMIZED UPGRADING OF INTERNET-ENABLED DEVICES BASED ON USER-PROFILE relates to a server system that maintains a user

profile of a particular end-user of consumer electronics network-enabled equipment and a data base of new technical features for this type of equipment. If there is a match between the user-profile and a new technical feature, and the user indicates to receive information about updates or sales offers, the user gets notified via the network of the option to obtain the feature.

U.S. serial no. 09/189,535 (attorney docket PHA 23,527) filed 11/10/98 for Eugene Shteyn for UPGRADING OF SYNERGETIC ASPECTS OF HOME NETWORKS relates to a server that has access to an inventory of devices and capabilities on a user's home network. The inventory is for example a look-up service as provided by HAVi or Jini architecture. The server has also access to a data base with information of features for a network. The server determines if the synergy of the apparatus present on the user's network can be enhanced based on the listing of the inventory and on the user's profile. If there are features that are relevant to the synergy, based on these criteria, the user gets notified.

U.S. patent 5,959,536 (attorney docket PHA 23,169) issued to Paul Chambers and Saurabh Srivastava for TASK-DRIVEN DISTRIBUTED MULTIMEDIA CONSUMER SYSTEM relates to a control system comprises multiple consumer electronics devices and task-driven control means coupled to the devices for controlling an interaction among the devices. The control means acts on respective software representations of each respective one of the consumer devices. By encapsulating the variable complexity of the task within a software representation, it can be made as simple or as sophisticated as needed to bring the capabilities up to a common level. Since the level of interface is common to the devices, applications can uniformly manipulate devices which embody very different levels of sophistication.

CLAIMS:

1. A method of enabling to bridge a first cluster of first functionalities and a second cluster of second functionalities in a home network, wherein:
- the first cluster has a first software architecture that provides control of the first functionalities through a programmatic interface;
- the second cluster has a second software architecture that provides control of the second functionalities through a UI-based interface; and
- the method comprises providing a translation of the programmatic interface to the UI-based interface.

2. The method of claim 1, wherein the first software architecture complies with UPnP, and the second software architecture complies with HAVi.

3. The method of claim 2, wherein the providing comprises generating a HAVi DDI target software component from a UPnP device description document.

4. A home network comprising:
- a first cluster of first functionalities; and
- a second cluster of second functionalities;
wherein:
- the first cluster has a first software architecture that provides control of the first functionalities through a programmatic interface;
- the second cluster has a second software architecture that provides control of the second functionalities through a UI-based interface; and
- the network has a bridge between the first and second clusters for translating the programmatic interface to the UI-based interface.

5. The network of claim 4, wherein the first software architecture complies with UPnP, and the second software architecture complies with HAVi.

6. The network of claim 5, comprising a generator for generating a HAVi DDI target software component from a UPnP device description document.

7. Bridging software for use on a home network, wherein:
- the network comprises a first cluster of first functionalities and a second cluster of second functionalities;
- the first cluster has a first software architecture that provides control of the first functionalities through a programmatic interface; and
- the second cluster having a second software architecture that provides control of the second functionalities through a UI-based interface;
- the bridging software is operative to couple the first and second clusters based on translating the programmatic interface to the UI-based interface.

8. The software of claim 7, for coupling the first software architecture that complies with UPnP with the second software architecture that complies with HAVi.

9. The software of claim 8, comprising a generator for generating a HAVi DDI target software component from a UPnP device description document.

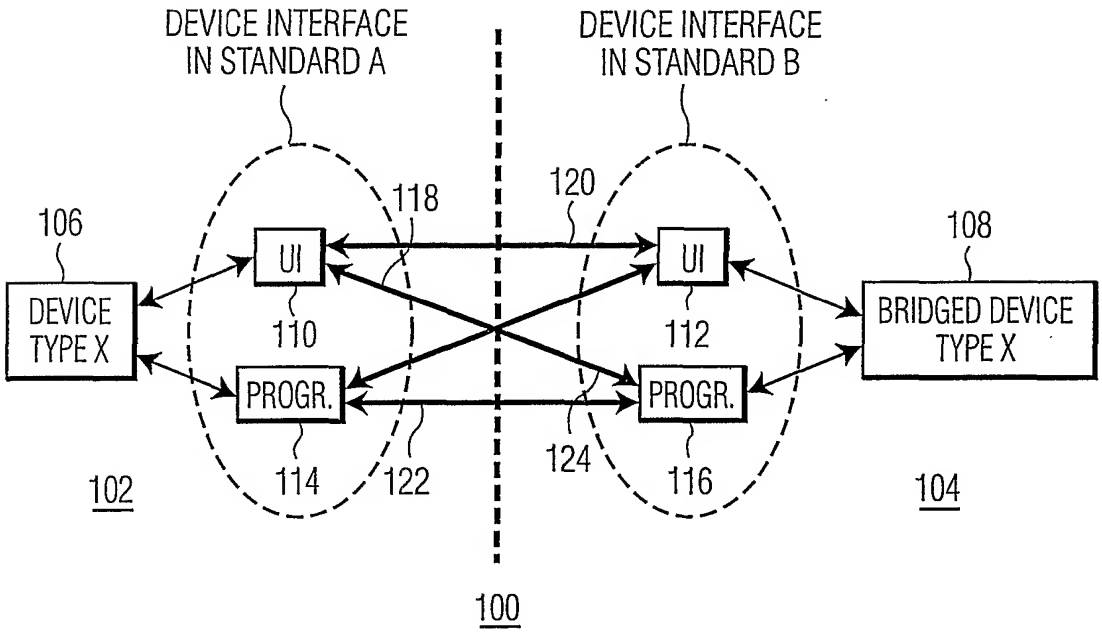
DEVICE INTERFACE
IN STANDARD A
DEVICE INTERFACE
IN STANDARD B
106
108
118
120
UI
UI
DEVICE
TYPE X
BRIDGED DEVICE
TYPE X
110
112
PROGR.
PROGR.
114
122
124
116
102
104
100

FIG. 1

UPnP DATA TYPE	ADD. SPECIFIERS	HA Vi DDI ELEMENT
BOOLEAN	N/A	DdiToggle, WITH FIELDS toggleName SET TO NAME OF STATE VARIABLE OR ACTION PARAMETER, numToggleStates SET TO 2. toggleStates SET TO 2 STATES, LABELED "TRUE" AND 'FALSE' OR 'ON' AND 'OFF' OR DdiChoice, WITH FIELDS choiceName SET TO NAME OF STATE VARIABLE OR ACTION PARAMETER, choiceType SET TO EQUAL choiceNumber set to 2 choiceList SET TO 2 STATES, LABELED 'TRUE' AND 'FALSE' OR 'ON' AND 'OFF'
ui1, ui2, ui4, i1, i2, i4, int	NONE SPECIFIED	DdiEntry, WITH FIELDS entryName SET TO NAME OF STATE VARIABLE OR ACTION PARAMETER entryType SET TO NAT_NUMBER
ui1, ui2, ui4, i1, i2, i4, int	<MINIMUM>, <MAXIMUM>, <STEP>	DdiSetRange, WITH FIELDS rangeName SET TO NAME OF STATE VARIABLE OR ACTION PARAMETER, valueRange SET TO THE RANGE SPECIFIED BY <MINIMUM> AND <MAXIMUM> stepValue SET TO VALUE OF <STEP> IN A STATE VARIABLE CONTEXT A DdiShowRange, WITH ANALOGOUS FIELDS, CAN BE USED INSTEAD OF A NON-INTERACTIVE DdiSetRange.
r4, r8, NUMBER, FIXED.14.4	NONE SPECIFIED	DdiEntry, WITH FIELDS entryName SET TO NAME OF STATE VARIABLE OR ACTION PARAMETER entryType SET TO FLOAT
r4, r8, NUMBER, FIXED.14.4	<MINIMUM>, <MAXIMUM>, <STEP>	DdiEntry, WITH FIELDS entryName SET TO NAME OF STATE VARIABLE OR ACTION PARAMETER entryType SET TO FLOAT (NOTE THAT DdiSetRange IS NOT DEFINED FOR FLOATING POINT VALUES).

FIG. 2A

STRING,CHAR	NONE SPECIFIED	DdiEntry, WITH FIELDS entryName SET TO NAME OF STATE VARIABLE OR ACTION PARAMETER entryType SET TO TEXTUAL
STRING,CHAR	<allowedValueList>	DdiChoice, WITH FIELDS choiceName SET TO NAME OF STATE VARIABLE OR ACTION PARAMETER, choiceType SET TO EQUAL choiceNumber SET TO THE NUMBER OF ALLOWED VALUES choiceList SET TO A SEQUENCE OF LENGTH 'choiceNumber', WHERE THE NAME OF choiceElement NUMBER n IS SET TO ALLOWED STRING VALUE NUMBER n, (0 <= n <choiceNumber)
DATE	N/A	DdiEntry, WITH FIELDS entryName SET TO NAME OF STATE VARIABLE OR ACTION PARAMETER entryType SET TO DATE
Time, time.tz	N/A	DdiEntry, WITH FIELDS entryName SET TO NAME OF STATE VARIABLE OR ACTION PARAMETER entryType SET TO TIME.
dateTime, dateTime.tz	N/A	TWO DdiEntry ELEMENTS, WITH FIELDS entryName SET TO THE NAME OF STATE VARIABLE OR ACTION PARAMETER AND entryType SET TO TIME FOR THE TIME PART AND DATE FOR THE DATE PART.
uri.	N/A	DdiEntry, WITH FIELDS entryName SET TO NAME OF STATE VARIABLE OR ACTION PARAMETER entryType SET TO TEXTUAL
bin.hex, bin.bin64, uuid	N/A	DdiEntry, WITH FIELDS entryName SET TO NAME OF STATE VARIABLE OR ACTION PARAMETER entryType SET TO TEXTUAL NOTE THAT HEXADECIMAL DATA IS PROBABLY NOT INTENDED FOR RAW PRESENTATION TO END-USERS. HENCE, IGNORING THIS TYPE OF DATA IS PROBABLY ACCEPTABLE AS WELL.

FIG. 2B

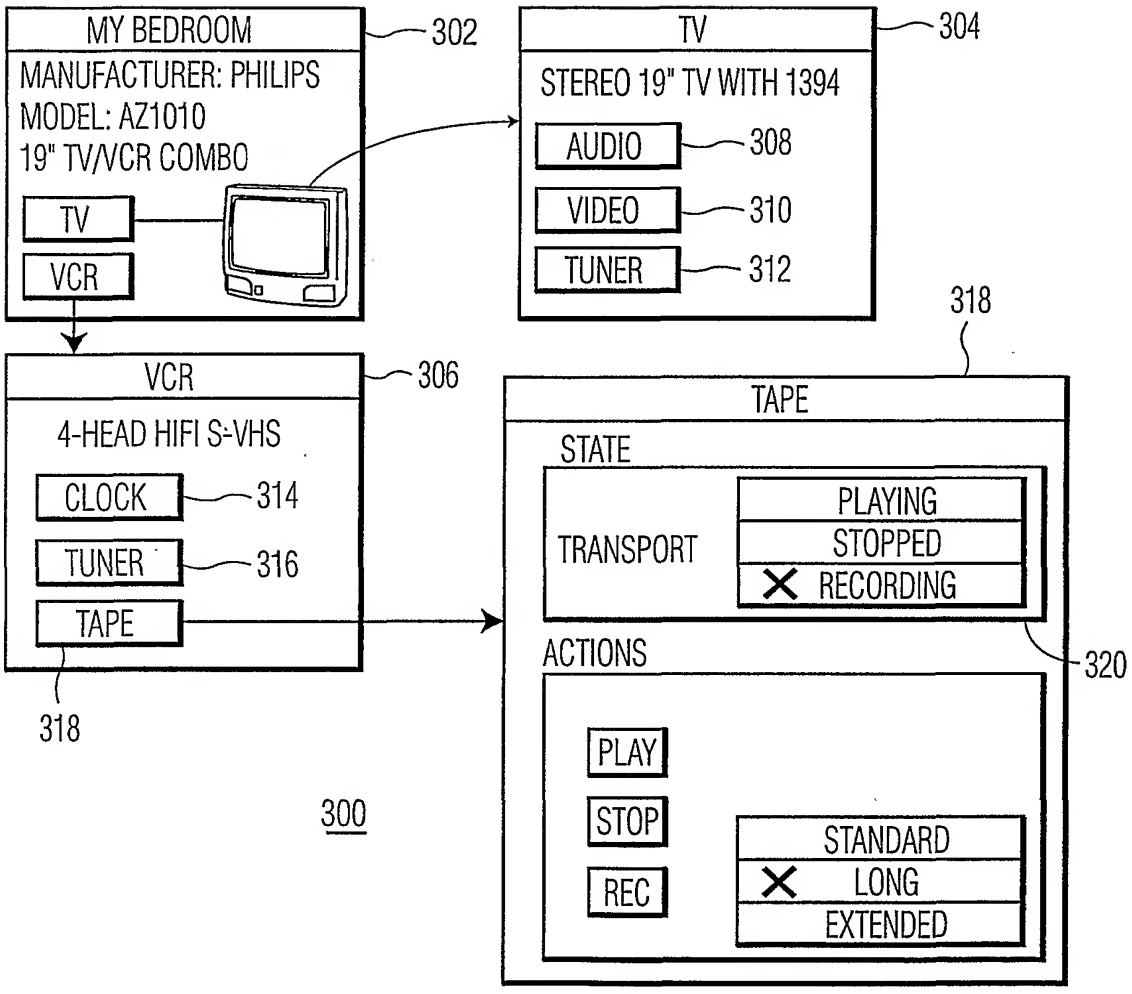
MY BEDROOM
302
MANUFACTURER: PHILIPS
MODEL: AZ1010
19" TV/VCR COMBO
TV
VCR
TV
304
STEREO 19" TV WITH 1394
AUDIO
308
VIDEO
310
TUNER
312
318
VCR
306
4-HEAD HIFI S-VHS
CLOCK
314
TUNER
316
TAPE
318
TAPE
STATE
TRANSPORT
PLAYING
STOPPED
RECORDING
320
ACTIONS
PLAY
STOP
REC
STANDARD
LONG
EXTENDED
300

FIG. 3

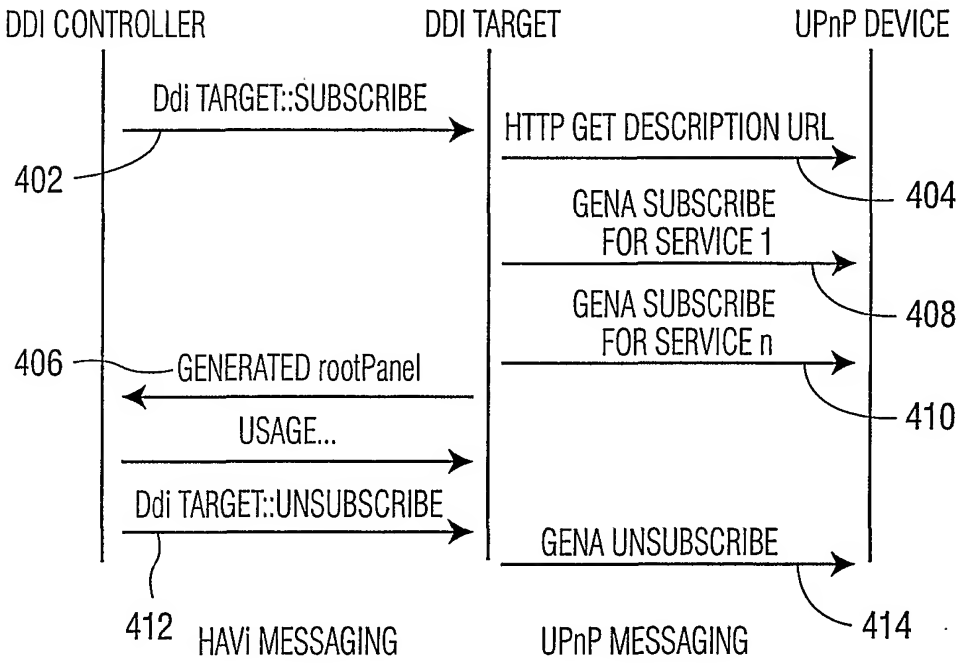
DDI CONTROLLER
DDI TARGET
UPnP DEVICE
Ddi TARGET::SUBSCRIBE
HTTP GET DESCRIPTION URL
402
404
GENA SUBSCRIBE FOR SERVICE 1
GENA SUBSCRIBE FOR SERVICE n
408
406
GENERATED rootPanel
410
USAGE...
Ddi TARGET::UNSUBSCRIBE
GENA UNSUBSCRIBE
412
HAVi MESSAGING
UPnP MESSAGING
414

FIG. 4

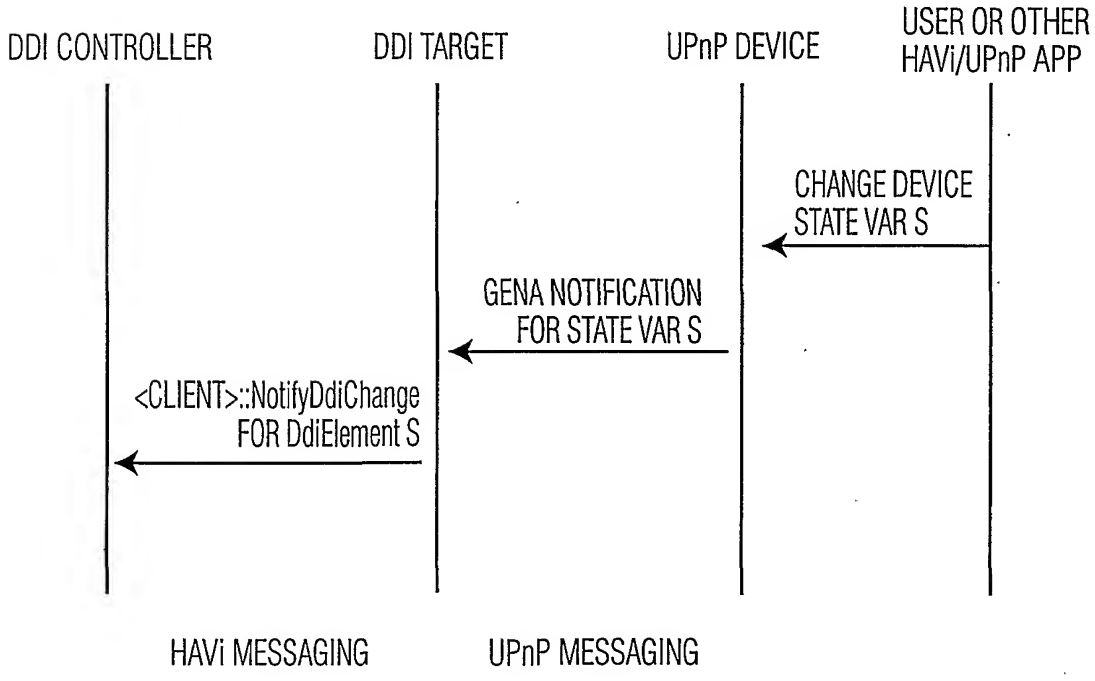
DDI CONTROLLER
DDI TARGET
UPnP DEVICE
USER OR OTHER HAVi/UPnP APP
CHANGE DEVICE STATE VAR S
GENA NOTIFICATION FOR STATE VAR S
<CLIENT>::NotifyDdiChange FOR DdiElement S
HAVi MESSAGING
UPnP MESSAGING

FIG. 5

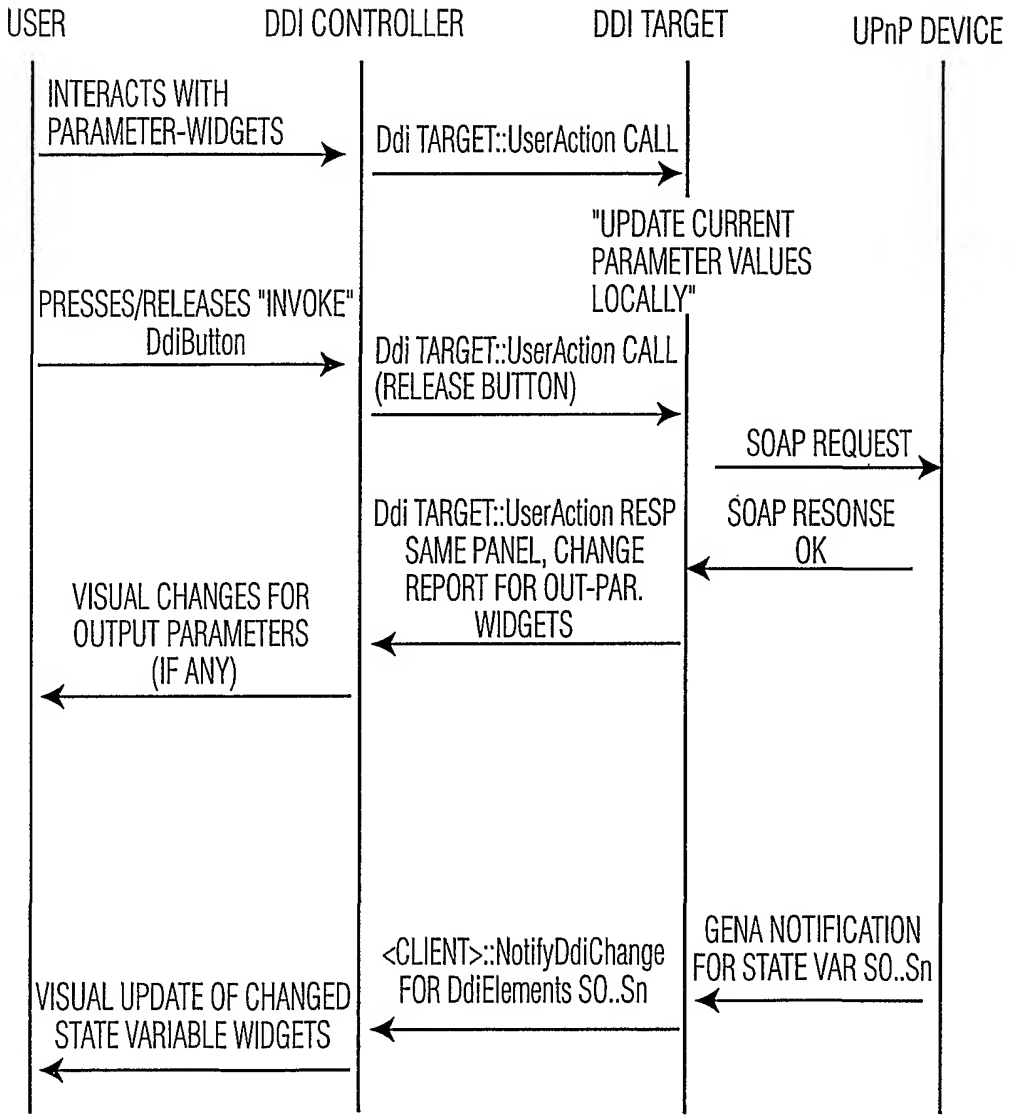
USER
DDI CONTROLLER
DDI TARGET
UPnP DEVICE
INTERACTS WITH PARAMETER-WIDGETS
Ddi TARGET::UserAction CALL
"UPDATE CURRENT PARAMETER VALUES LOCALLY"
PRESSES/RELEASES "INVOKE" DdiButton
Ddi TARGET::UserAction CALL (RELEASE BUTTON)
SOAP REQUEST
SOAP RESONSE OK
Ddi TARGET::UserAction RESP SAME PANEL, CHANGE REPORT FOR OUT-PAR. WIDGETS
VISUAL CHANGES FOR OUTPUT PARAMETERS (IF ANY)
GENA NOTIFICATION FOR STATE VAR S0..Sn
<CLIENT>::NotifyDdiChange FOR DdiElements S0..Sn
VISUAL UPDATE OF CHANGED STATE VARIABLE WIDGETS

FIG. 6

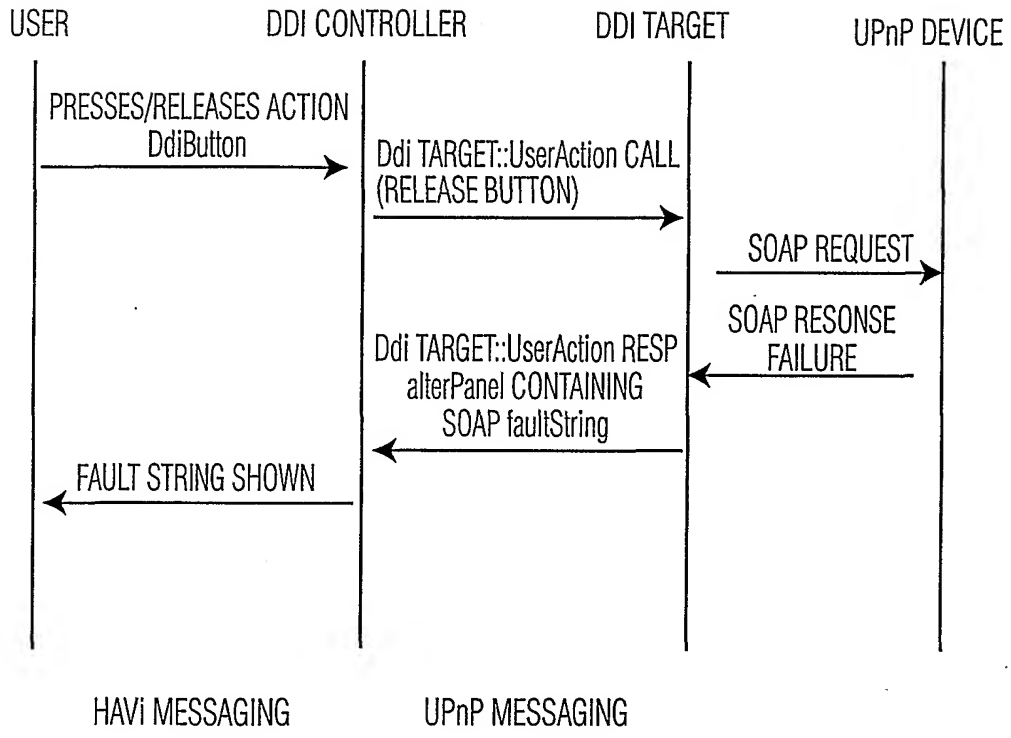
USER
DDI CONTROLLER
DDI TARGET
UPnP DEVICE
PRESSES/RELEASES ACTION
DdiButton
Ddi TARGET::UserAction CALL
(RELEASE BUTTON)
SOAP REQUEST
SOAP RESONSE
FAILURE
Ddi TARGET::UserAction RESP
alterPanel CONTAINING
SOAP faultString
FAULT STRING SHOWN
HAVi MESSAGING
UPnP MESSAGING

FIG. 7